Dicembre 1998

F. Spagna

I/O E ASPETTI AVANZATI DELLA TECNOLOGIA JAVA

F. Spagna

I/O E ASPETTI AVANZATI DELLA TECNOLOGIA JAVA

5. Aspetti avanzati di Java

5.1 Programmazione concorrente

5.1.1 Programmazione concorrente e thread

Il linguaggio Java è predisposto per la **programmazione concorrente** mediante *thread* (programmazione *multithreaded*) che permette ad un programma di poter svolgere al suo interno in modo efficiente più compiti simultaneamente in parallelo indipendenti l'uno dall'altro (concorrenti), come ad esempio la presentazione di un'animazione grafica, la contemporanea riproduzione di un suono e la sorveglianza di certi eventi nello stesso tempo, oppure lo svolgimento di un compito di fondo (per esempio un calcolo pesante) contemporaneamente ad altre attività di un programma. In un programma che si svolge in un unico thread le operazioni che hanno tendenza ad impegnare molto il tempo del processore, come i loop di lunga durata o senza fine, possono portare a situazioni di blocco per le quali il processore, totalmente occupato, diventa indisponibile per altri compiti e risponde male agli eventi: si può allora limitare l'assorbimento di risorse di sistema di queste operazioni facendole agire e regolamentandole nell'ambito di un thread. E' consigliabile utilizzare un thread in quei casi in cui in un programma viene svolto un processamento continuo, come in un'animazione.

Il multithreading permette ad esempio la contemporanea esistenza di diverse applet sulla stessa pagina che girano indipendentemente l'una dall'altra.

Un **thread** è un singolo flusso sequenziale di esecuzione all'interno di un processo (programma), con un avvio, una sequenza di esecuzione ed una fine, il quale può agire contemporaneamente ad altri thread concorrenti che svolgono compiti diversi. Il thread è diverso dal *processo* in quanto, mentre due processi hanno ciascuno il proprio spazio di memoria indirizzata (o area di dati), due thread, che fanno parte dello stesso processo, possono condividere lo stesso spazio di indirizzi del processo, le stesse variabili globali, i file aperti e altre cose, e tra di loro non c'è protezione, anche se ciascuno di essi si riserva alcune risorse proprie. Per questo i thread sono anche detti *processi leggeri (lightweight)*.

In Java per far funzionare una classe sotto forma di thread si può adoperare uno dei due modi che descriviamo qui di seguito.

Un ***primo modo*** di creare un thread consiste nella definizione di una classe come ***sottoclasse della classe*** **`Thread`** (del package `java.lang`):

```
class miaClasThr extends Thread {
   public void run() {
      // istruzioni del metodo
   }
}
```

ridefinendovi il metodo `run()`, ereditato dalla superclasse `Thread`, nel corpo del quale viene inserito tutto il codice relativo alla sequenza di esecuzione del thread. Si crea quindi un'istanza della classe così definita e la si avvia come thread mediante il metodo `start()` ereditato anch'esso da `Thread`:

```
miaClasThr mioThr = new miaClasThr();
mioThr.start();
```

Infatti il metodo `run()` posseduto da ogni oggetto di classe `Thread` (e quindi anche da ogni classe da questa derivata), che è un metodo in cui si svolge tutta l'attività del thread, viene lanciato automaticamente dal metodo `start()` di `Thread` alla fine dell'esecuzione di questo, e, quando il `run()` finisce, il thread si estingue. Il metodo `run()` è eseguito in background e la classe intanto può proseguire la sua esecuzione. Un thread può essere terminato anche con un'esplicita chiamata del suo metodo `stop()`.

Esempio dal JDK stesso:#

Esempio1#

Un ***secondo modo*** per creare un thread è quello di creare una classe implementandola con l'interfaccia **`Runnable`** e ridefinire in essa, come questa interfaccia richiede, il suo unico metodo `run()`, inserendovi tutto il codice proprio di esecuzione del thread:

```
class miaClas implements Runnable {
   miaClas() { }                    // mettere il costruttore?#
   public void run() {
      // implementazione del metodo run()
   }
}
```

Se allora si crea un thread come istanza della classe `Thread` dandogli nel costruttore come argomento un oggetto della classe precedentemente definita (`Thread` ha un costruttore che riceve come argomento un oggetto che implementa `Runnable`), gli viene così attribuito il metodo `run()` di questa, che viene eseguito automaticamente dopo il metodo `start()` del thread:

```
miaClas mioOggRun = new miaClas();
Thread mioTh = new Thread(mioOggRun);
mioTh.start();    // oppure semplicemente new Thread(mioOggRun).start();
```

Esempio2 Esempio di JDK#

Ma non è detto che il thread debba essere esterno alla classe, esso può anche essere creato nell'ambito della classe stessa come un suo membro (variabile d'istanza):

```
class miaClas implements Runnable {
   Thread mioTh;
   public void run() {
      // implementazione del metodo run()
   }
```

```
}
```

che viene istanziato in un opportuno metodo della classe, dandogli questa volta come argomento nel costruttore la variabile `this` (cioè l'oggetto stesso della classe):

```
Thread mioTh = new Thread(this);
```

Esempio3:#

Questo secondo modo di creare un thread da una classe implementata come `Runnable`, piuttosto che derivarla dalla classe `Thread`, viene generalmente applicato quando si vuole ridefinire soltanto il metodo `run()` del `Thread` e non altri, mentre la ridefinizione di altri metodi del `Thread` può essere fatta solo con il primo modo.

Per ogni thread si può stabilire una priorità ed anche un tempo durante il quale il thread lascia libero il campo agli altri thread concorrenti, con il metodo:

```
mioTh.sleep(100);
```

avente il tempo (in millisecondi) come parametro. Lo `sleep()` imposto ad un thread ogni tanto, per esempio ad ogni ciclo di un loop, è importante per lasciare tempo al processore di eseguire altri thread. Ma quando si vuole veramente sospendere l'esecuzione di un thread in attesa di certi eventi deve essere adoperato il metodo `suspend()` e riprenderla con il metodo `resume()`. Si fa qui un esempio di sospensione e ripresa di un thread con comando mediante click del mouse in relazione al valore di una variabile d'istanza booleana che abbiamo chiamato per esempio `sospeso`:

```
public boolean mouseDown(Event e, int x, int y) {
   if (sospeso) mioTh.resume();
   else         mioTh.suspend();
   sospeso = !sospeso;
   return true;
}
```

Nel caso di un'applet che si voglia fare funzionare in un thread (è spesso consigliabile farlo), è l'applet stessa (naturalmente derivata dalla classe `Applet`) che viene implementata come `Runnable`, e il thread viene creato come un'oggetto di classe `Thread` facente parte dell'applet stessa in qualità di suo membro:

```
public class miaApp extends    Applet
                    implements Runnable {
   Thread mioTh = null;    // dichiarazione di referenza ad un oggetto Thread
   // altre variabili e metodi dell'applet
}
```

In questo caso l'istanziazione del thread e la chiamata al suo metodo `start()` saranno fatte nel metodo `start()` dell'applet, che, come è spiegato al paragrafo 4.7.3.2, è attivato dal sistema ogni volta che l'applet prende visibilità sul documento HTML. Ma è anche nel metodo `start()` dell'applet che si crea un nuovo `Thread` con il `this` (cioè l'applet stessa) come argomento del costruttore, se non ne è stato ancora creato uno o se l'istanza del thread precedente è terminata (nei due casi il `Thread` ha valore `null`):

```
public void start() {
```

```
    if (mioTh == null) {
       mioTh = new Thread(this);
       mioTh.start();
    }
}
```

Quando l'applet termina perché la pagina HTML è abbandonata è eseguito automaticamente il suo metodo `stop()` (quello della classe `Applet`), e quindi in questo metodo dell'applet si deve fermare con il suo metodo `stop()` (quello della classe `Thread`) l'esecuzione del thread, che altrimenti continuerebbe ad agire anche a pagina non più presente, ed assegnare al thread il valore `null`, il che lo rende disponibile alla *garbage collection*: se ne potrà poi creare uno nuovo ad una nuova esecuzione (automatica) del metodo `start()` quando la pagina dell'applet viene ripresa:

```
public void stop() {
    if (mioTh != null) {
       mioTh.stop();
       mioTh = null;   // perche' possa ripartire con il prossimo start()
    }
}
```

Il codice relativo all'esecuzione dell'applet viene invece implementato nel metodo `run()`. Si riporta qui di seguito un esempio con un loop che ridisegna ciclicamente l'applet (con un `repaint()`) e si arresta quando il thread finisce, cioè quando esso diventa uguale a `null`:

```
public void run() {
    while (mioTh != null) {             // fintantochè c'è il thread
             try {
       Thread.sleep(100);             // thread in attesa per 100 msec
             } catch (InterruptedException e) { }
       repaint();                      // ridisegna l'area dell'applet
    }
}
```

Poichè il metodo `sleep()` mette in attesa il thread per un certo numero di millisecondi (100 nell'esempio) viene permesso in questo tempo ad altri thread concorrenti di agire. Il tempo sarà stabilito facendo un compromesso tra le esigenze legate alla velocità di *refreshing* dell'azione contenuta nel `while` e quelle di allungare al massimo i tempi resi disponibili per altri thread.

La sospensione dell'esecuzione di un qualsiasi programma per un determinato tempo (n millisecondi) può essere fatta con il metodo di classe, chiamato senza istanziazione di alcun oggetto `Thread`:

```
Thread.sleep(n);
```

anche in caso di programmazione non concorrente.

Nuove frasi:#

Ogni thread ha una sua priorità rispetto agli altri thread.

Quando una Java Virtual Machine parte funziona in un thread che è quello che chiama il metodo `main()` della classe.

5.1.2 Esempio di programma con thread

Si riporta qui di seguito un caso generale, con le operazioni più comuni sui thread, come dichiarazione, creazione `new Thread(...)`, `start()`, `sleep(...)`, `resume()` e `suspend()`: lo schermo viene rinnovato ogni 100 millisecondi con la scritta di un numero via via crescente (con sospensione o riavvio ad ogni clic del mouse sull'applet). Lo schema è applicabile ad ogni altro caso in cui si voglia ridisegnare continuamente un'applet, cambiando solo il metodo `paint()`.

```
// E01casoTh.java (F.Spagna) Esempio generale di Thread

import java.awt.Graphics;

public class E01casoTh extends java.applet.Applet
                       implements Runnable {              // interfaccia Runnable
   Thread Th = null;                                     // thread non ancora creato
   boolean sospeso = false;  // variabile che indica la sospensione del thread
   int i;                          // variabile d'istanza contatore di cicli
   public void start() {
      if (Th == null) {           // se thread non ancora creato o annullato
         Th = new Thread(this);                        // il thread si crea adesso
         Th.start();                                                // e lo si avvia
      }
   }
   public void stop() {
      Th.stop();
      Th = null;                 // se causato stop(), thread viene annullato
   }
   public void run() {             // metodo che è eseguito in continuazione
      while (Th != null) {                           // fintantochè c'è il thread
         try { Thread.sleep(100); }          // thread in attesa per 100 msec
         catch (InterruptedException e){}
         repaint();                                           // ridisegna l'area
      }
   }
   public void paint(Graphics g) {
      g.drawString("i = " + i++, 10, 10);
   }
   public boolean mouseDown(java.awt.Event evt, int x, int y) {
      if (sospeso)  Th.resume();
      else          Th.suspend();
      sospeso = !sospeso;
      return true;
   }
}
```

5.1.3 Interfaccia `Runnable`

L'interfaccia **`Runnable`** facente parte del package `java.lang`, contiene il solo metodo astratto **`run()`** che deve essere implementato, come è richiesto per i metodi di ogni interfaccia, all'atto della definizione della classe cui questa interfaccia viene applicata.

5.1.4 Programmazione *thread-safe*

Quando in un programma dei dati in un oggetto possono essere modificati contemporaneamente da threads diversi viene richiesta una programmazione thread-safe mediante blocchi di codice sincronizzati (*synchronized*) nei quali un solo thread alla volta può agire mentre gli altri aspettano o con il meccanismo di wait/notify.

5.2 Serializzazione degli oggetti

La serializzazione degli oggetti estende le classi di input/output di Java al supporto degli oggetti.

Esempio di programma che prima memorizza scrivendolo su un file un oggetto di classe `String` serializzato e poi lo rilegge da quel file e ne presenta la stringa sullo schermo.

```
// E02serializ043.java (F.Spagna) Esempio di serializzazione

import java.io.*;

public class E02serializ043 {

   public static void main(String args[]) throws IOException {

      String s = "0123456789 fer ";
      FileOutputStream fos = new FileOutputStream("out");
      ObjectOutputStream oos = new ObjectOutputStream(fos);
      oos.writeObject(s);
      oos.flush();
      fos.close();

      FileInputStream fis = new FileInputStream("out");
      ObjectInputStream ois = new ObjectInputStream(fis);
                  try {
      String ss = (String)ois.readObject();
      fis.close();
      System.out.print(ss);
                  } catch (ClassNotFoundException e) {}
   }
}
```

Esempio con immagine#

5.3 Programmazione di rete

Java, a differenza degli altri linguaggi di programmazione che l'hanno preceduto, è stato già alla sua nascita concepito con l'obiettivo, tra gli altri fondamentali, di una sua utilizzazione in applicazioni di rete e esso rappresenta oggi il migliore strumento per questo tipo di programmazione, dove le sue caratteristiche di indipendenza dalla piattaforma, sicurezza e internazionalità dei caratteri, ne fanno un linguaggio particolarmente adatto ad Internet, per cui questi aspetti sono fondamentali. Con Java la programmazione di rete è resa molto più semplice rispetto ad altri linguaggi e il codice relativo alla rete si riduce spesso a poche istruzioni.

In una comunicazione tra un client Java ed un server (in particolare tra un' applet ed una servlet) possono essere scmbiati tra client e server dati di qualsiasi tipo, contrariamente a quello che è permesso da un browser Web in una connessione HTTP, che ammette solo alcuni formati: un client può inviare dati (stringhe di query) raccolti dall'utente tramite un modulo (form) e il server può restituire file come testo o file HTML e alcuni formati di immagini visualizzabili dal browser. A questo si aggiunga il fatto che l'interfaccia Java sul client può presentarsi con un aspetto grafico evoluto e permette, tramite l'applicazione client, di fare ogni tipo di trattamento e di presentazione dei dati, uscendo dai limiti ristretti del formato HTML (pur eventualmente arricchito con Javascript): poichè la libreria grafica AWT di Java ha tutti i controlli disponibili nei form, e anzi anche molto di più, si possono progettare dei form intelligenti e interattivi che alla fine inviano i dati al server simulandone il formato HTTP. E' anche possibile che la connessione con il server sia aperta in qualunque momento automaticamente a programma, periodicamenre o in seguito a qualche evento, per ricevere dal server (o inviare ad esso) in tempo reale dei dati poi localmente elaborati ed i cui risultati sono presentati sul client.

Java permette di scrivere dei *"protocol handler"* per comunicare con diversi tipi di server e dei *"content handler"* per interpretare e presentare diversi tipi di dati.

Le applet, che, per le restrizioni di sicurezza imposte dai browser, non sono autorizzate ad accedere al disco locale per scrivervi su file, possono però memorizzare dei dati su un server tramite CGI o servlet. Così, tramite un server che funziona da intermediario, è possibile anche aggirare il divieto per un'applet di aprire una connessione con un sito diverso da quello del server che l'ha inviata e fare dialogare diverse applet in rete, e ciò apre la porta ad applicazioni collaborative di rete, giochi a più giocatori, chat (scambio di messagi in tempo reale tra diversi client), whiteboard (scambio di immagini disegnate in tempo reale).

[] Elliotte Rusty Harold, Programmare in rete con Java (traduzione dall'inglese Network Java Programming, 1997), O'Reilly, Jackson Libri, 1998.

Java permette i collegamenti in rete di applet e applicazioni mediante classi contenute nel package `java.net`. Tali classi offrono funzionalità di rete indipendenti dalla piattaforma secondo i **protocolli Web** oppure **socket tipo Unix**.

Le applet non permettono, per ragioni di sicurezza, la lettura e la scrittura di file sulla macchina sulla quale sono eseguite (client) e non possono aprire connessioni con sistemi diversi da quello da cui provengono.

5.3.1 Classe **URL**

La classe **URL** rappresenta un indirizzo di tipo URL. Un **URL** (**U**niform **R**esource **L**ocator) è una risorsa sul Web che può essere un file, una directory, o anche una query. Nella sua forma generale un URL è costituito da varie parti: il protocollo, il nome della macchina host (server), l'eventuale numero di porta, la directory ed il nome di file, che può essere un file presente sul server o anche un programma eseguibile sul server in modalità CGI che produce un file (o meglio un flusso di dati o stream, per esempio formattati html) al volo. Tutti questi elementi di un URL sono definiti come stringhe membri privati della classe `URL`: `protocol`, `host`, `port`, `file`, `ref`. Un esempio potrebbe essere:

`http://www.sito.it:8080/direct/file.html#paragrafo`

Un URL può specificare un numero di porta (quello della connessione TCP del server) preceduto da due punti. Quando il numero di porta non è specificato viene assunta quella di default, che per il protocollo http è la 80.

L'URL può anche essere relativo alla pagina corrente.

I costruttori della classe sono:

`URL(String stringaURL)`
crea un oggetto `URL` da un'unica stringa di URL comprendente:
protocollo (`http://`), nome del server (`www.server.it/`) e nome del file (`file.html`)
richiede il `catch` di un'eccezione di tipo `MalformedURLException`

`URL(String protocol, String host, String file)`
come sopra ma con tre argomenti separati

`URL(String protocol, String host, int port, String file)`
come sopra con il numero di porta in più

`URL(URL url, String file)`
crea un `URL` da un altro `URL` e in più un nome di file (con path)

Metodi principali:

`URLConnection openConnection()`
restituisce una `URLConnection` che rappresenta una connessione all'`URL` stesso

`InputStream openStream()`
apre una connessione all'`URL` e ne restituisce un `InputStream` per leggervi dei dati
equivale a: `openConnection().getInputStream()`

La classe **URL** ha il metodo `openStream()` che permette di aprire una connessione di rete su un determinato `URL` e restituisce un'istanza della classe `InputStream` (del package `java.io`) sul quale si possono leggere dei dati.

`showDocument()` permette ad un'applet di far caricare al browser un'altra pagina Web

```
getAppletContext.showDocument();
```

5.3.2 Connessione URL

Esempio:

(v.pag.293) pari pari (è come quella che ho già fatto)#

try

InputStream is = url.openStream();

DataInputStream dis = new DataInputStream(is);

catch(IOException)

5.3.3 Classe `URLconnection`

La classe astratta **`URLConnection`** serve per rappresentare una connessione di comunicazione con un URL, che consente di leggere o scrivere nella risorsa rappresentata dall'URL.

Un'istanza di questa classe è creata invocando il metodo `openConnection()` di un oggetto URL.

`connect()`	permette di aprire una connessione di comunicazione alla risorsa dell'URL cui si riferisce l'`URLConnection`

5.3.4 Invio di un messaggio (`mailto`)

Mediante un URL, la sua connessione e il relativo OutputStream è possibile inviare un messaggio di posta elettronica (e-mail). Viene qui di seguito fatto un esempio con un'applet avente una serie di campi di input per preparare il messaggio ed un bottone che produce l'invio del messaggio:

```
// E03mail60.java (F.Spagna) Invio di una e-mail con l'URL mailto:

import java.io.* ;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Date;

public class E03mail060 extends java.applet.Applet {

   TextField TFhost;
   TextField TFfrom;
   TextField TFto;
   TextField TFsubj;
   TextArea  TFmess;
   Button    Binvio;
```

```
   Button    Bchiud;
   Label     Lavviso;

   public void init() {

      add(new Label("Host:"));  add(TFhost = new TextField(35));
      add(new Label("From:"));  add(TFfrom = new TextField(35));
      add(new Label("To:   "));  add(TFto   = new TextField(35));
      add(new Label("Subj:"));  add(TFsubj = new TextField(35));
      add(new Label("Mess:"));  add(TFmess = new TextArea(8, 35));
      add(Binvio = new Button("invio"));
      add(Bchiud = new Button("chiudi"));
      add(Lavviso = new Label("in attesa di invio"));
      Lavviso.setForeground(Color.red);
      setFont(new Font("Arial", Font.BOLD, 14));
   }

   public boolean action(Event evt, Object arg) {

      if (arg.toString().equals("chiudi"))
         System.exit(0);                                  // se bottone chiusura
      else if (arg.toString().equals("invio")) {                 // chiude tutto
                  try {                                    // se bottone invio
         System.getProperties().put("mail.host", TFhost.getText());
         String from      = TFfrom.getText();
         String to        =   TFto.getText();
         String subject   = TFsubj.getText();
         String messaggio = TFmess.getText();
         Lavviso.setText("connessione");
         URL u = new URL("mailto:" + to);                        // crea il mailto
         URLConnection c = u.openConnection();          // crea una URLConnection
         OutputStream os = c.getOutputStream();//ne ricava lo stream di output
         PrintWriter out = new PrintWriter(new OutputStreamWriter(os));
         c.setDoInput(false);                           // non input a questo url
         c.setDoOutput(true);                                       // output si
         c.connect();                                // connessione al mail host
         out.println("From:     " + from       + "\n" +
                     "To:       " + to         + "\n" +
                     "Date:     " + new Date() + "\n" +
                     "Subject: " + subject    + "\n" +
                     "\n"         + messaggio); //riga bianca dopo intestazione
         Lavviso.setText("messaggio inviato");
         out.close();
                                                        // termina il messaggio
                  } catch (Exception e) { Lavviso.setText("errore"); }
      }
      return true;
   }
   public static void main(String[] args) {

      Frame fr = new Frame("mail060");
      mail060 m60 = new mail060();
      fr.add(m60, "Center");
      m60.init();
      m60.start();
      fr.resize(400, 360);
      fr.show();
   }
}
```

L'aspetto dell'applet viene riportato nella figura 5.11.
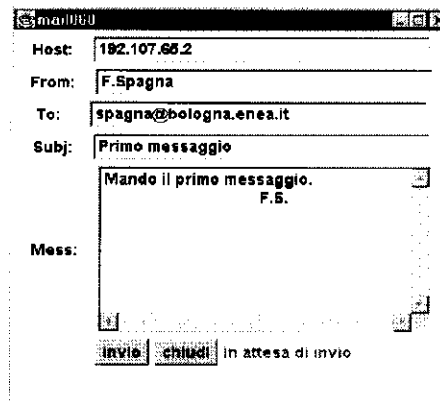
Figura 5.1 Applet per l'invio di un messaggio e-mail.

5.3.5 Socket

Un **socket** è un punto di attacco (*endpoint*) per la comunicazione via rete tra due macchine.

Le classi del package `java.net` della libreria standard di Java supportano i socket del tipo BSD (Berkeley...).

Le classi `Socket` e `ServerSocket` permettono di aprire connessioni socket standard tra client e server e leggere e scrivere su di essi.

5.3.5.1 Classe `Socket`

Costruttori:

`Socket()`
crea un socket non connesso con il tipo di SocketImpl di default del sistema

`Socket(InetAddress, int)`
crea un socket stream e lo connette ad un dato numero di porta ad un dato indirizzo IP

`Socket(InetAddress, int, InetAddress, int)`
crea un socket e lo connette ad un dato remote address su un dato remote port

`Socket(SocketImpl)`
crea un socket non connesso con a SocketImpl dato dall'utente

`Socket(String, int)`
crea un socket stream e lo connette ad un dato numero di porta su un dato host

`Socket(String, int, InetAddress, int)`
crea un socket e lo connette ad un dato remote host su un dato remote port

Metodi:

`close()`	chiude il socket
`getInetAddress()`	restituisce l'address al quale il socket è connesso

`getInputStream()`	restituisce uno stream di input per il socket
`getLocalAddress()`	restituisce il local address al quale il socket è bound
`getLocalPort()`	restituisce il local port al quale il socket è bound
`getOutputStream()`	restituisce uno stream di output per il socket

5.3.5.2 Connessioni socket

Le classi **`Socket`** e **`ServerSocket`** permettono di aprire connessioni in rete socket standard tra client e server e leggere e scrivere su di essi, con protocolli diversi da quelli possibili con le classi `URL` e `URLConnection` e offrono maggiori possibilità per applicazioni di rete più generali con tecniche di tipo socket Unix standard nella quale quindi un client e un server comunicano tra di loro in rete.

Per aprire una connessione socket si crea un'istanza della classe **`Socket`**:

```
Socket sk = new Socket(nomeHost, numPorta);
```

Per leggere o scrivere in questa connessione si possono allora usare gli stream di input e output di Java, `DataInputStream` e `DataOutputStream`, che si ottengono con i metodi `getInputStream()` e `getOutputStream()` del socket.

copia pagina 296 Lemay sotto#

e alla fine bisogna chiudere la connessione socket:

```
sk.close();
```

il che chiude anche gli InputStream e OutputStream collegati ad esso.

Dal lato server ci deve essere un socket server sotto forma di un oggetto di tipo **`ServerSocket`** che resta in attesa su una porta TCP di una connessione richiesta da un cliente, quando questo si connette al server va in azione il metodo **`accept()`**, che permette l'ascolto di richieste client su quella porta e l'accettazione della connessione.

Per creare un socket di tipo server si crea un'istanza della classe `ServerSocket` con un costruttore:

```
ServerSocket ssk = new ServerSocket(8000);
```

che richiede come argomento il numero di porta.

Una volta che la connessione è stabilita il server per leggere e scrivere sul socket con il client usa gli stream di input e di output di questo.

Si riporta qui un esempio con il codice relativo ad un server ed un client che comunicano mediante socket:

```
// E04server048.java (C.Poli, F.Spagna) Esempio di socket server

import java.net.ServerSocket;
```

```java
import java.net.Socket;
import java.io.*;

public class E04server048 {

   ServerSocket ss;                                    // socket del server
   Socket cs;                                          // socket del client

   E04server048() {
                        try {
      ss = new ServerSocket(1328);
      System.out.println("il server e' partito e resta in attesa");
      do {
         cs = ss.accept();                             // resta in ascolto
         System.out.println("cliente: " + cs.getInetAddress());
         InputStream is = cs.getInputStream();
         DataInputStream dis = new DataInputStream(is);
         OutputStream os = cs.getOutputStream();
         PrintStream ps = new PrintStream(os);
         String s = "";
         do {
            s = dis.readLine();
            System.out.println("il cliente dice: " + s);
            ps.println("ricevuto: " + s);
         } while (!s.equals("quit"));
         cs.close();
      } while (true);
      // ss.close();
                        } catch(IOException e) {}
   }
   public static void main(String s[]) {
      new E04server048();
   }
}
```

```java
// E05client048.java (C.Poli, F.Spagna) Esempio di socket client

import java.net.Socket;
import java.io.*;

public class E05client048 {

   Socket cs;                                          // socket del client
   String ip = "rin365.arcoveggio.enea.it";

   E05client048() {
                        try {
      cs = new Socket(ip, 1328);
      System.out.println("aperta la connessione con " + ip);
      InputStream is = cs.getInputStream();
      OutputStream os = cs.getOutputStream();
      DataInputStream dis = new DataInputStream(is);
      PrintStream ps = new PrintStream(os);
      String s = "";
      do {
         s = new DataInputStream(System.in).readLine();
         ps.println(s);
         System.out.println(dis.readLine());
      } while (!s.equals("quit"));
      cs.close();
                        } catch(IOException e) {}
   }
   public static void main(String s[]) {
```

```
        new E05client048();
    }
}
```

Con le classi dell'esempio precedente il server può avere un solo collegamento alla volta, e per poter invece avere diversi collegamenti contemporaneamente il programma va modificato con l'uso di thread: il server istanzia un thread per ogni nuova connessione attraverso una classe specifica. Qui di seguito sono presentate le classi relative, cominciando da quella del server:

```
// E06server048.java (C.Poli, F.Spagna)
// Server socket Versione per diversi clienti con thread

import java.net.ServerSocket;
import java.net.Socket;
import java.io.*;

public class E06server049 {

   E06server049() {
                        try {
      ServerSocket ss = new ServerSocket(1328);
      while (true) { new conness(ss.accept()); }          // resta in ascolto
                        } catch(IOException e) {}
   }
   public static void main(String s[]) {

      new E06server049();
   }
}

class conness extends Thread {

   static int nCon = 0;
   static int totCon = 0;
   Socket cs;
   DataInputStream dis;
   PrintStream ps;
   String ip;

   conness(Socket cs) {

            try {
      this.cs = cs;
      ip = cs.getInetAddress().toString();
      System.out.println("cliente: " + ip + " (connessione " + ++nCon + ")");
      dis = new DataInputStream(cs.getInputStream());
      ps = new PrintStream(cs.getOutputStream());
      ps.println("conn. n. " + nCon + " (clienti attivi " + ++totCon + ")");
            } catch(IOException e) {}
      start();
   }
   public void run() {
            try {
      String s = "";
      do {
         s = dis.readLine();
         System.out.println("conness. " + nCon + " (" + ip + ") dice: " + s);
         ps.println("ricevuto: " + s);
      } while (!s.equals("quit"));
      cs.close();
      totCon--;
```

```
        System.out.println("cliente " + ip + " scollegato");
              } catch(IOException e) {}
   }
}
```

Anche la classe cliente va riscritta di conseguenza:

```
//  E07client049.java (C.Poli, F.Spagna) Socket
//  Socket client versione per diversi clienti con thread

import java.net.Socket;
import java.io.*;

public class E07client049 {

   Socket cs;
   String ip = "rin365.arcoveggio.enea.it";

   E07client049() {
            try {
      cs = new Socket(ip, 1328);
      System.out.println("aperta la connessione con " + ip);
      InputStream is = cs.getInputStream();
      OutputStream os = cs.getOutputStream();
      DataInputStream dis = new DataInputStream(is);
      PrintStream ps = new PrintStream(os);
      System.out.println(dis.readLine());
      String s = "";
      do {
         s = new DataInputStream(System.in).readLine();
         ps.println(s);
         System.out.println(dis.readLine());
      } while (!s.equals("quit"));
      cs.close();
            } catch(IOException e) {}
   }
   public static void main(String s[]) {

      new E07client049();
   }
}
```

5.4 Utilizzazione di codice nativo in Java (JNI)

Nelle applicazioni Java è possibile utilizzare del codice nativo proprio di una certa piattaforma scritto usando altri linguaggi di programmazione. Una tale esigenza può essere dovuta a varie ragioni tra cui principalmente: l'utilizzo di funzionalità proprie di una certa piattaforma non previste per ragioni di portabilità in Java, l'utilizzazione di codice preesistente (*legacy*) scritto in altri linguaggi e ben funzionante e motivi di prestazioni migliori che con Java. Questa possibilità può facilitare la transazione verso Java.

Con il JDK 1.1 è stata introdotta la **Java Native Interface (JNI)**.

Nella definizione di una classe Java può essere indicato che un metodo è implementato a parte in modo nativo mediante il modificatore `native`. L'applicazione `javah.exe` applicata al codice Java crea un file header in C che contiene i prototipi delle funzioni native che implementano i metodi della classe dichiarati `native`.

5.5 Eccezioni

Le **eccezioni** sono delle ... di errori del programma. Java ha un sistema di **gestione delle eccezioni** incluso nel suo sistema di run-time.

La sintassi dell'istruzione `try` ... `catch`... è la seguente:

```
try {      /* blocco di codice in cui intercettare possibili errori */ }
catch (tipoDiEccezione1 e) { /* istruzioni trattamento eccezione 1 */ }
catch (tipoDiEccezione2 e) { /* istruzioni trattamento eccezione 2 */ }
catch (tipoDiEccezione3 e) { /* istruzioni trattamento eccezione 3 */ }
// . . .
finally { /* facoltativo: istruzioni trattamento ogni altra eccezione */ }
}
```

Quando si produce un errore all'interno del blocco **`try`** il blocco stesso viene abbandonato, viene "rilevata un'eccezione" ed è creata un'istanza della classe dell'eccezione relativa. Il tipo di tale oggetto è poi esaminato dalle istruzioni **`catch`** che seguono, nelle quali ogni clausola `catch` prende in considerazione un tipo diverso di eccezione possibile. e viene eseguito il blocco `catch` relativo a quel tipo di eccezione. Il **`finally`**, che è facoltativo, va in esecuzione per ogni altra eccezione non considerata specificamente.

Se nessuna eccezione si presenta nel blocco `try`, il blocco `finally`, se presente, viene comunque eseguito ed il programma prosegue. La presa in conto delle eccezioni è obbligatoria per tutta una serie di istruzioni particolari che la esigono, ma non è obbligatorio il loro trattamento nei vari blocchi `catch`, che possono anche essere vuoti.

Delle eccezioni possono essere anche sollevate da un'istruzione **`throw`**.

Le undici eccezioni possibili in Java sono:

`ArithmeticException`	divisione (o modulo intero) per zero
`NoClassDefFoundException`	riferimento ad una classe non definita

IncompatibleClassChangeException	cambiamento illegale di definizione di una classe
OutOfMemoryException	memoria insufficiente per la creazione di un oggetto
NullPointerException	tentativo di accesso ad un oggetto vuoto
ClassCastException	casting illegale su un oggetto
ArrayIndexOutOfBoundsException	tentativo di accesso fuori dai limiti di un array
NegativeArraySizeException	tentativo di creare un array a lunghezza negativa
IncompatibleTypeException	tentativo di istanziare un'interfaccia
UnsatisfiedLinkException	metodo nativo che non può essere linkato
InternalException	errore interno del sistema di run-time

2. Input e output (I/O) in Java

2.1 Package `java.io`

2.1.1 *Stream* di input e output e classi relative

Gli input e gli ouput in Java vengono gestiti attraverso quelli che si chiamano ***stream*** (flussi di dati).

Le classi `InputStream` e `OutputStream` sono due classi astratte che rappresentano un flusso di dati rispettivamente in lettura e in scrittura. Esse danno origine per subclassing a tutta una serie di classi Java standard specializzate per vari tipi di lettura o di scrittura.

Quando si ha una connessione URL rappresentata da un oggetto di tipo `URLConnection` oppure un socket rappresentato da un'istanza della classe `Socket`, i due tipi di stream che leggono da e scrivono su quelle connessioni aperte possono essere ottenuti rispettivamente con il metodo `getInputStream()` e `getOutputStream` che sia la classe `URLConnection` sia quella `Socket` (così come anche la classe `Process` che rappresenta un processo) posseggono (si vedano i paragrafi 5.3.1, 5.3.4 e 6.5.1.3). Inoltre la classe `URL` ha il metodo `openStream()` che apre una connessione ad essa e ne restituisce lo stream di input.

2.1.2 Classe InputStream

La classe astratta **`InputStream`** è la superclasse di tutte le classi che rappresentano un flusso (*stream*) di byte di input. Le sottoclassi che vengono derivate da essa devono sempre definire un metodo `read()`.

Metodi principali:

`int read()`	legge e restituisce il prossimo byte nei dati del flusso come un `int` (si noti bene questo punto!) di un byte (valore compreso tra 0 e 255): se non si trova più alcun byte da leggere (fine dello stream) il metodo restituisce -1.
`read(byte[])`	legge un certo numero di byte e li immagazzina in un buffer

Altri metodi:

`skip(long n)`	salta e ignora nella lettura un numero n di byte
`close()`	chiude il flusso di dati di input e rilascia ogni risorsa di sistema da esso impegnata
`mark(int)`	marca la posizione corrente di lettura nel flusso di dati letti cui si riposizionerà la lettura se viene in seguito chiamato il metodo `reset()`
`reset()`	riprende la posizione di lettura precedentemente fissata con il metodo

```
mark()
```

La classe `InputStream` ha un solo metodo di lettura `read()` che può leggere solo dei `byte`, singoli o in array. Maggiori possibilità di lettura in diversi formati sono offerte invece dalla classe `DataInputStream`: ed è perciò che il più delle volte per la lettura viene usato un oggetto di questa classe.

2.1.3 Stream di uscita di un nuovo processo aperto

Si fa adesso un esempio in cui l'output di un programma lanciato da Java viene mostrato sullo schermo:

```
// F01run.java (F.Spagna) Mostra sul monitor l'output di un programma

import java.io.*;

public class F01run {
   public static void main(String args[]) throws Exception {
      Process p = Runtime.getRuntime().exec(args[0]);
      InputStream is = p.getInputStream();
      byte b[] = new byte[4096];
      int c;
      while ((c = is.read(b)) != -1)    // legge b.length byte e li stocca in b
         System.out.write(b, 0, c);      // write(byte[] b, int off, int len)
      p.destroy();
   }
}
```

Una volta questo programma compilato si può provarlo ad esempio con il comando:

```
java F09run "java F07type F09run.java"
```

con l'`F07type` compilato del paragrafo 6.5.1.4.#

L'intercettazione da parte di un'applicazione Java dell'output di un altro programma si presta in qualche caso al riutilizzo in Java di qualche semplice programma tradizionale (*legacy*). Qui si riporta solo un esempio di un programma Java che si limita a riscrivere l'uscita di un eseguibile compilato in qualsiasi linguaggio (piccole differenze si manifestano tra C, Fortran e Basic) sullo schermo (dopo averla messa tutta in caratteri maiuscoli per fare un esempio di trattamento) e anche slvarla su file. Ecco il codice:

```
// F02run089.java (F.Spagna) Utilizza in Java l'output di un altro programma
// 01-08.06.99 (inizio 8 giugno 1999)

import java.io.*;

public class F02run089 {
   public static void main(String args[]) throws Exception {
      Process p = Runtime.getRuntime().exec(args[0]);   // apre nuovo processo
      InputStream is = p.getInputStream();              // output del processo
      FileOutputStream fos = new FileOutputStream("fil.out");//riceve l'output
```

```
      byte b[] = new byte[4096];        // array di byte per contenere l'output
      int c = 0;                             // numero di byte letti ogni volta
      int totale = 0;                             // numero totale di byte letti
      while ((c = is.read(b)) != -1) { // legge b.length byte e li stocca in b
         System.out.write(b, 0, c);        // write(byte[] b, int off, int len)
         fos.write(b, 0, c);                      // scrive l'output nel file
         totale += c;
      }
      String s = new String(b, 0, totale);  // dai byte costruisce una stringa
      System.out.println(s.substring(0, totale).toUpperCase());  // la elabora
      fos.close();
      p.destroy();
   }
}
```

Un esempio un po' più elaborato che lancia il comando DOS "chkdsk", ne preleva l'output e lo inserisce in un file di formato pdf che visibilizza con l'Acrobat Reader è il seguente:

```
// F03dospdf090.java (F.Spagna) Utilizza in Java l'output di un altroprogramma
// 01-08.06.99 (inizio 8 giugno 1999)

import java.awt.*;
import java.io.*;
import uk.org.retep.pdf.*;

public class F03dospdf090 {

   String s;
   F03dospdf090() throws Exception {
      Process p = Runtime.getRuntime().exec("chkdsk");  // apre nuovo processo
      InputStream is = p.getInputStream();               // output del processo
      byte b[] = new byte[4096];        // array di byte per contenere l'output
      int c = 0;                             // numero di byte letti ogni volta
      int totale = 0;                             // numero totale di byte letti
      while ((c = is.read(b)) != -1) { // legge b.length byte e li stocca in b
         System.out.write(b, 0, c);        // write(byte[] b, int off, int len)
         totale += c;
      }
      s = new String(b, 0, totale);          // dai byte costruisce una stringa
      p.destroy();
      String comando = "c:/Acrobat 4.0/Reader/Acrord32.exe pagina.pdf";
      Runtime.getRuntime().exec(comando);                 // apre nuovo processo
                                 try {
      String fil = "pagina.pdf";
      FileOutputStream fos = new FileOutputStream(fil);
      PDFJob job = PDF.getPDFJob(fos);
      Graphics g = job.getGraphics(PDFPage.LANDSCAPE);
      g.drawLine(10, 60, 770, 60);
      g.setColor(new Color(240, 240 ,255));
      g.fillRect(10, 75, 770, 310);
      g.setColor(Color.red);
      g.setFont(new Font("SansSerif", Font.BOLD, 24));
      g.drawString("chkdsk", 40, 45);
      g.setFont(new Font("Monospaced", Font.BOLD, 16));
      for (int n = 0; n < 14; n++) {
         g.drawString(s.substring(0, s.indexOf('\n')), 20, 100+20*n);
         s = s.substring(s.indexOf('\n')+1);
      }
      g.dispose();
      job.end();                                          // termina il job
```

```
        fos.close();
                                    } catch(Exception e) {
                                      System.out.println("Errore " + e);
                                      e.printStackTrace(); }
    }
    public static void main(String args[]) throws Exception {
        new F03dospdf090();
    }
}
```

La pagina pdf è visualizzata dall'Acrobat Reader nel modo della figura 6.3.

chkdsk

```
Numero di serie del volume: 0916-12FC

2.550.669.312 byte di spazio totale su disco
1.268.285.440 byte disponibili su disco

        4.096 byte in ogni unit  di allocazione
      622.722 unit  di allocazione su disco
      309.640 unit  di allocazione disponibili su disco

      655.360 byte di memoria complessiva
      569.536 byte disponibili

Invece di utilizzare CHKDSK, provare con SCANDISK. SCANDISK  in grado di
identificare e risolvere un pi  grande numero di problemi del disco.
```

Figura 6.3 Output di un comando DOS intercettato e portato su un file pdf.

2.1.4 Classe OutputStream

La classe astratta **`OutputStream`** è la superclasse di tutte le classi che rappresentano un flusso (*stream*) di byte di output. Alle sottoclassi che vengono derivate da essa è sempre richiesto di definire un metodo `write()`:

Metodi principali:

`write(int b)`	scrive come `int` un byte nei dati del flusso
`write(byte[])`	scrive i byte di un array sul flusso di output

Altri metodi:

`flush()`	svuota il flusso di output e forza alla scrittura ogni buffer di byte di output
`close()`	chiude il flusso di dati di output e rilascia ogni risorsa di sistema da esso impegnata

2.1.5 Classi FileReader, PrintReader e FileWriter

La classe **`FileReader`** a#

Esempio di programma che legge un file di testo e lo mostra sullo schermo:

```
// F04type.java (F.Spagna) Mostra sul monitor il contenuto di un file

import java.io.*;

public class F04type {
   public static void main(String args[]) throws Exception {
      FileReader fr = new FileReader(args[0]);
      PrintWriter pw = new PrintWriter(System.out, true);
      char c[] = new char[4096];
      int  read = 0;
      while ((read = fr.read(c)) != -1)                // fino all'end of file
         pw.write(c, 0, read);
      fr.close();
      pw.close();
   }
}
```

Esempio di programma che legge un file di testo e lo copia in un altro file:

```
// F05copy.java (F.Spagna) Copia un file in un altro

import java.io.*;

public class F05copy {

   public static void main(String args[]) throws Exception {
      if (args.length < 2) {
         System.err.println("uso: java F08copy InputFile OutputFile");
         System.exit(1);
      }
      FileReader fr = new FileReader(args[0]);
      FileWriter fw = new FileWriter(args[1]);
      char c[] = new char[4096];
      int read = 0, total = 0;
      while ((read = fr.read(c)) != -1) {              // fino all'end of file
         fw.write(c, 0, read);
         total += read;
      }
      fr.close();
      fw.close();
      System.out.println(total + " byte copiati");
   }
}
```

2.1.6 Classe **File**

La classe **File** rappresenta un file o una directory del file system della macchina su cui il programma gira.

Costruttori:

File(String path)	crea un file con un dato path
File(String path, String nome)	crea un file con un dato path e un dato nome
File(File dir, String nome)	crea un file con una data directory e un dato nome

Alcuni metodi:

isFile()	restituisce `true` se si tratta di un file
isDirectory()	restituisce `true` se si tratta di una directory
exists()	restituisce `true` se il file esiste
list()	restituisce una lista di file contenuti nella directory data

Esempio di un programma che riceve una stringa in input da linea di comando e scrive tutti i file, le directory e le sottodirectory a partire dalla directory rappresentata da quella stringa:

```
// F06filesist.java (F.Spagna) Struttura del file system
// 01-24.11.98 (inizio 24 novembre 1998)

import java.io.*;

public class F06filesist {

   public static void dir(File f) {
      System.out.println(f);                            // scrive dir o file
      if (f.isDirectory()) {                             // se e' directory
         String fil[] = f.list();                     // vede come e' composta
         for (int i = 0; i < fil.length; i++)               // e per ognuno
            dir(new File(f, fil[i]));               // ripete ricorsivamente
      }
   }
   public static void main(String args[]) {
      File f = new File(args[0]);
      if (f.exists())                 // analizza la stringa immessa in input
         dir(f);                                // se e' buona ne fa il dir
      else                                                   // altrimenti
         System.err.println("non accessibile:" + f);     // segnala l'errore
   }
}
```

Segue un esempio che applica vari metodi della classe `File`:

```
// F07file.java (F.Spagna) Metodi della classe File

import java.io.*;

public class F07file {

   public static void main(String s[]) throws IOException {

      File f = new File("c:\\f\\javdoc\\file.java");
      File d = new File("c:\\f\\javdoc");
      System.out.println(
           "\n f isFile()              " + f.isFile()
         + "\n f isDirectory           " + f.isDirectory()
         + "\n d isFile()              " + d.isFile()
         + "\n d isDirectory()         " + d.isDirectory()
         + "\n separator               " + f.separator
         + "\n pathSeparator           " + f.pathSeparator
         + "\n f getName()             " + f.getName()
         + "\n d getName()             " + d.getName()
         + "\n getPath()               " + d.getPath()
         + "\n getAbsolutePath()       " + f.getAbsolutePath()
         + "\n isAbsolute()            " + f.isAbsolute()
         + "\n getCanonicalPath()      " + f.getCanonicalPath()
         + "\n getParent()             " + f.getParent()
         + "\n exists()                " + f.exists()
         + "\n canWrite()              " + f.canWrite()
         + "\n canRead()               " + f.canRead()
         + "\n lastModified()          " + (new java.util.Date(f.lastModified()))
         + "\n length()                " + f.length()
      );
      for (int n = 0; n < d.list().length; n++)
         System.out.println(" list()[" + n + "]   " + d.list()[n]);
   }
}
```

La schermata risultante è riportata nella figura 6.4.

```
f isFile()              true
f isDirectory           false
d isFile()              false
d isDirectory()         true
separator               \
pathSeparator           ;
f getName()             file.java
d getName()             javdoc
getPath()               c:\f\javdoc
getAbsolutePath()       c:\f\javdoc\file.java
isAbsolute()            true
getCanonicalPath()      c:\F\javdoc\file.java
getParent()             c:\f\javdoc
exists()                true
canWrite()              true
canRead()               true
lastModified()          Sun May 02 01:41:34 PDT 1999
length()                1427
list()[0]   file.java
list()[1]   file.class
list()[2]   colori.html
list()[3]   sorgenti
```

Figura 6.4 Applicazione di vari metodi della classe `File`.

2.1.7 Lettura dati da file e scrittura su file

E' scritto qui di seguito un programma che legge i dati da un file (`fil.in`) e li scrive su un altro file (`fil.out`).

```
// F08ioFile.java (F.Spagna) Lettura e scrittura su file

import java.io.*;

public class F08datiFile {

   public static void main(String args[]) {
              try {
      FileInputStream fis = new FileInputStream("fil.in");
      DataInputStream dis = new DataInputStream(fis);
      String s = dis.readLine();                    // riceve la stringa da file

      FileOutputStream fos = new FileOutputStream("fil.out");
      PrintStream ps = new PrintStream(fos);
      ps.println(s);
              } catch (java.io.FileNotFoundException e) {
                   System.out.println("file non esistente");
              } catch (java.io.IOException e) {
                   System.out.println("errore di input/output"); }
   }
}
```

Per scrivere dei dati su file si usa un oggetto (noi lo abbiamo chiamato `ps`) della classe `PrintStream` che ha un metodo `println()`, proprio come quella dell'oggetto `out` anch'esso della classe `PrintStream`, della classe `System` del paragrafo 3.2.1, collegandolo ad un oggetto (chiamato da noi `fos`) della classe `FileOutputStream` inizializzato col nome del file su cui si vuole scrivere.

Per ottimizzare la lettura da un file è bene leggere il maggior numero possibile di dati contemporaneamente con una sola istruzione, per esempio per leggere una sequenza di byte, piuttosto che leggerli uno alla volta, è preferibile leggere un ... e mettere in un buffer che poi viene...#

2.1.8 Lettura dati di un file remoto

Esempio di apertura di una connessione e di lettura da un file posto su una macchina remota.

```
// F09letRemot.java  F.Spagna  Lettura da un file remoto

import java.io.*;
import java.net.*;

public class F09letRemot {

   public static void main(String args[]) {
              try {
```

```
            URL url = new URL("http://nettuno.it");
            URLConnection c = url.openConnection();
            DataInputStream dis = new DataInputStream(c.getInputStream());
            String s = "";
            while ((s = dis.readLine()) != null)
               System.out.println(s);
                     } catch(MalformedURLException e) { }
                       catch(java.io.IOException e) {}
    }
}
```

Nel caso sia un'applet a leggere un file ricordiamo che essa può leggere solo file che siano posti sullo stesso server da cui essa arriva, ed in tal caso il codice sarebbe:

```
// F10lettrice055.java (F.Spagna) Applet che legge un file dal suo server
import java.awt.*;
import java.io.*;
import java.net.*;

public class F10lettrice055 extends java.applet.Applet {

   String riga;                           // riga del file letta di volta in volta
   String testo = "";                            // testo completo del file letto
   TextArea ta;                                        // area di testo scorrevole

   public void init() {
          try {                                       // metodo che parte all'inizio
      add(ta = new TextArea(10, 50));                // aggiunge un'area di testo
      ta.setFont(new Font("Arial", Font.BOLD, 16));              // fissa il font
      URL url = getDocumentBase();           // URL completo della stessa applet
      URLConnection c = url.openConnection();                 // apre connessione
      DataInputStream dis = new DataInputStream(c.getInputStream());
      while ((riga = dis.readLine()) != null)                 // ciclo di lettura
         testo = testo + riga + "\n";   // appende la riga al testo precedente
          } catch(MalformedURLException e) { }
            catch(java.io.IOException e) { }
      ta.setText(testo);
   }                                           // mette il testo nella sua area
}
```

E il risultato sarebbe quello di figura 6.5.
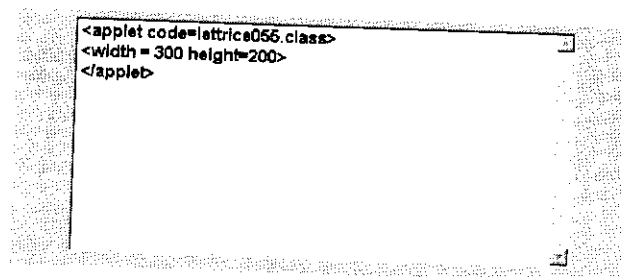
Figura 6.5 Risultato grafico dell'applet che legge lo stesso file HTML che la richiama.

2.1.9 Input da tastiera e output sullo schermo

E' riportato qui un programma che chiede all'utente una stringa da tastiera, la riceve e la presenta sullo schermo, poi rifà la stessa operazione in un modo diverso, e infine riceve da tastiera un numero e lo scrive sullo schermo.

I dati da tastiera sono sempre immessi come `byte` o `String`: per immettere dei numeri si introduce la stringa che li rappresenta che è letta e trasformata in valore numerico nel programma, come è stato fatto nelle ultime tre righe di codice nell'esempio.

```
// F11datiConsole.java (F.Spagna) Lettura da tastiera e scrittura su schermo

import java.io.*;

public class F11datiConsole {

   public static void main(String args[]) {
                                                                    // scrive invito
      System.out.println("dai il nome");

                                                           // nome come array di byte
      byte nome[] = new byte[20];                        // riceve il nome da tastiera
      int n = System.in.read(nome);                            // lo scrive su schermo
      System.out.write(nome, 0, n);

      DataInputStream dis = new DataInputStream(System.in);
      String s = dis.readLine();                      // riceve la stringa da tastiera
      System.out.println(s);                                  // la scrive su schermo

      String sn = in.readLine();                      // riceve la stringa da tastiera
      int numero = Integer.valueOf(sn).intValue();     // la trasforma in numero
      System.out.println(numero);                           // lo scrive sullo schermo
   }
}
```

Il metodo `System.in.read()` legge tutti i caratteri che l'utente ha immesso sulla tastiera fino a che non viene battuto il tasto di invio e li inserisce nell'array di byte, potendo l'utente fare anche correzioni col tasto di ritorno indietro (*backspace*) sulla riga prima dell'invio. Se l'utente oltrepassa il numero di caratteri per cui l'array è stato dimensionato, il metodo non tiene conto di quelli in eccesso.

Altro esempio che scrive sullo schermo uno alla volta i caratteri via via che vengono battuti sulla tastiera, come se si trattasse di una macchina da scrivere:

```
// F12dattilo.java (F.Spagna) Input/output tastiera/schermo

import java.io.*;

public class F12dattilo {

   static final int ENTER = 13;
   public static void main(String args[]) {
      int x = 0;                         // carattere come numero (valore del tasto)
          try {
      while ((x = System.in.read()) != ENTER)      // finche' tasto non e' ENTER
         System.out.write((char)x);       // lo scrive come carattere su schermo
          } catch(java.io.IOException e) {} // obbligatorio per read, write
   }
```

```
}
```

2.1.10 Classe BufferedReader

La classe **BufferedReader**, che estende la classe Reader, legge un testo da uno stream di input di caratteri, ponendo i caratteri in un buffer (le cui dimensioni possono essere assegnate o essere quelle di default, sufficientemente grandi per la maggior parte dei casi) per una lettura efficiente di caratteri, array e righe.

In generale si avvolge un `BufferedReader` intorno a ciascun tipo di `Reader`, come i `FileReader` e gli `InputStreamReader` per renderne più efficienti le operazioni di lettura con `read()`. Per esempio:

```
BufferedReader br = new BufferedReader(new FileReader("testo.txt"));
```

produce la bufferizzazione dell'input dal file specificato.

Tale classe può essere utilizzata per la lettura di testi al posto della `DataInputStream` usata con il JDK 1.0.

Costruttori:

BufferedReader(Reader)
crea uno stream di input a carattere con un buffer di input con dimensioni di default

BufferedReader(Reader, int)
crea uno stream di input a carattere con un buffer di input di dimensioni date

Metodi:

close()	chiude lo stream
mark(int)	marca la posizione attuale di lettura nello stream
markSupported()	dice se lo stream supporta l'operazione di `mark()`
read()	legge un singolo carattere
read(char[],int,int)	legge dei caratteri in una porzione di un array
readLine()	legge una riga di testo
ready()	dice se lo stream è pronto ad essere letto
reset()	riposiziona la lettura dello stream all'ultima marcatura
skip(long)	salta e ignora nella lettura un dato numero di caratteri

Esempio di lettura di un file di testo con `FileReader`:

```
// F13bufread.java (F.Spagna) Esempio di BufferReader per lettura filedi testo
import java.io.*;

public class F13bufread {

   public static void main(String[] s) {
            try {
      StringBuffer sb = new StringBuffer();
      BufferedReader br = new BufferedReader(new FileReader("CB.HTM"));
      String riga = br.readLine();
      while (riga != null) {
         sb.append(riga + "\n");
         riga = br.readLine();
      }
      System.out.println(sb);
            } catch (IOException e) {
               System.out.println("Errore di I/O : " + e.getMessage()); }
   }
}
```

Altro esempio di lettura di un input stream relativo a un file cui si accede tramite l'URL con `InputStreamReader`:

```
// F14bufread.java (F.Spagna) Esempio di BufferReader per lettura file ditesto
import java.io.*;

public class F14bufread {

   public static void main(String[] s) {
            try {
      StringBuffer sb = new StringBuffer();
      url = new URL("http://spagna/Essai.txt");
      InputStream is = url.openStream();
      BufferedReader br = new BufferedReader(new InputStreamReader(is));
      String riga = br.readLine();
      while (riga != null) {
         sb.append(riga + "\n");
         riga = br.readLine();
      }
      System.out.println(sb);
            } catch (IOException e) {
               System.out.println("Errore di I/O : " + e.getMessage()); }
   }
}
```

Serializzazione di un oggetto:

```
// F15serializ.java (F.Spagna) Serializzazione ridotta all'essenziale
import java.io.*;

public class F15serializ {

   public static void main(String s[])
```

```
                throws IOException, ClassNotFoundException {
        new ObjectOutputStream(new FileOutputStream("fil")).writeObject("ciao");
        Object o = new ObjectInputStream(new FileInputStream("f")).readObject();
        System.out.println(o + " - " + o.getClass().getName());
    }
}
```

Risultato:

```
ciao - java.lang.String
```

2.1.11 Classe BufferedWriter

La classe **BufferedWriter**, che estende la classe `Writer`, scrive un testo su uno stream di output di caratteri, ponendo i caratteri in un buffer (le cui dimensioni possono essere assegnate o essere quelle di default, sufficientemente grandi per la maggior parte dei casi) per una scrittura efficiente di caratteri, array e stringhe.

In generale si avvolge un `BufferedWriter` intorno a ciascun tipo di `Writer`, come il `FileWriter` o l'`OutputStreamWriter` per renderne più efficienti le operazioni di scrittura con `write()`. Per esempio:

```
BufferedWriter bw = new BufferedWriter(new FileWriter("testo.txt"));
```

produce la bufferizzazione dell'output sul file specificato.

Costruttori:

BufferedWriter(Writer)
______crea uno stream di output a carattere con un buffer di output a dimensioni di default

BufferedWriter(Writer, int)
______crea uno stream di output a carattere con un buffer di output di dimensioni date

Metodi:

`close()`	chiude lo stream
`flush()`	fa il "flush" dello stream (cioè lo svuota)
`newLine()`	scrive un separatore di fine riga: è preferibile usare questo metodo piuttosto che scrivere un "\n" che non è riconosciuto da tutte le piattaforme
`write(int)`	scrive un singolo carattere dato con il suo valore intero

write(char[],int,int)	scrive dei caratteri ponendoli in una parte di un array
write(String,int,int)	scrive una porzione di una stringa a partire dal carattere in una certa posizione e per una certa lunghezza

Esempio di scrittura di un file di testo con `FileWriter`:

```
// F16bufwrite.java (F.Spagna) Es.di BufferWriter per scriver su file di testo

import java.io.*;

public class F16bufwrite {

   public static void main(String[] s) {
            try {
      StringBuffer sb = new StringBuffer();
      BufferedReader br = new BufferedWriter(new FileWriter("CB.HTM"));
      String riga = br.readLine();
      while (riga != null) {
         sb.append(riga + "\n");
         riga = br.readLine();
      }
      System.out.println(sb);
            } catch (IOException e) {
               System.out.println("Errore di I/O : " + e.getMessage()); }
   }
}
```

2.2 Output su file PDF

La libreria **retepPDF** permette in un'applicazione Java l'uscita dei dati di una pagina grafica su un file di formato PDF (Portable Document Format) leggibile e stampabile con l'applicativo Adobe Acrobat Reader. La libreria non sopporta ancora la serializzazione e le immagini. La libreria è disponibile come file precompilato jar (**retepPDF.jar**) al sito http://www.retep.org.uk/.

Facciamo qui un esempio di una pagina grafica presentata con orientamento di tipo "*portrait*" (pagina verticale) e con orientamento "*landscape*" (pagina orizzontale):

```
// F17pdf088.java (F.Spagna) Output su un file PDF con la libreria RETEPPDF
// 04-04.06.99 (inizio: 3 giugno 1999)

import java.awt.*;
import java.io.*;
import uk.org.retep.pdf.*;

public class F17pdf088 {

   public F09pdf088() {
                                          try {
      String fil = "pagina.pdf";
      FileOutputStream fos = new FileOutputStream(fil);
      PDFJob job = PDF.getPDFJob(fos);
      paginaPortrait(job);                              // crea la pagina portrait
      paginaLandscape(job);                                     // pagina landscape
      job.end();                                                   // termina il job
      System.exit(0);                     } catch(Exception e) {
                                              System.out.println("Errore " + e);
                                              e.printStackTrace(); }
   }
   public void paginaPortrait(PDFJob job) {
      Graphics g = job.getGraphics(); //ricava un object Graphics del PrintJob
                                                 // cio' crea una nuova pagina
      g.setFont(new Font("SansSerif", Font.PLAIN, 12));
      g.drawString("Pagina in Portrait", 50, 50);
      scritte(g);
      disegni(g);
      g.dispose();                                   // getta l'oggetto graphics
   }
   public void paginaLandscape(PDFJob job) {
      Graphics g = job.getGraphics(PDFPage.LANDSCAPE);
      g.setFont(new Font("SansSerif", Font.PLAIN, 12));
      g.drawString("Pagina in Landscape", 50, 50);
      disegni(g);
      scritte(g);
      g.dispose();
   }
   public void disegni(Graphics g) {
      g.drawLine(0, 60, 400, 60);
      g.setColor(new Color(230, 230, 255));
      g.fillRect(40, 75, 260, 72);
      g.setColor(Color.black);
      g.drawRect(40, 75, 260, 72);
```

```
      int xp[] = new int[] {  80, 120, 160, 120, 80 };
      int yp[] = new int[] { 180, 180, 220, 220, 180 };
      int np = xp.length;
      g.setColor(Color.orange);
      g.fillPolygon(xp, yp, np);
      g.setColor(Color.black);
      g.drawPolygon(xp, yp, np);
      for (int n = 0; n < 36; n++) {
         g.setColor(new Color(n*6, (int)(Math.random()*255), 255 - n*6));
         g.fillArc(300, 150, 100, 100, n*10, 180);
      }
      g.setColor(Color.black);
      g.setFont(new Font("SansSerif", Font.PLAIN, 16));
      g.drawString("Pagina di prova retepPDF ", 300, 20);
      g.setFont(new Font("SansSerif", Font.PLAIN, 12));
      int y = 100;
      g.drawString("Documento prodotto con la libreria retepPDF ", 50, y);
      y += 12;
      g.drawString("Test v 1.3 1999/01/20 peter", 50, y);
      y += 12;
      g.drawString("Vers. JVM: " + System.getProperty("java.version"), 50, y);
      y += 12;
      g.drawString("Subpackage usato: " + PDF.getBasePackage(), 50, y);
   }
   public void scritte(Graphics g) {
      String fonti[] = new String[] {
         "SansSerif", "Monospaced", "TimesRoman",
         "Helvetica","Courier","Dialog","DialogInput"};
      String modi[] = new String[] {
         "Plain", "Bold", "Italic", "Bold+Italic"};
      int imodi[] = new int[] {Font.PLAIN, Font.BOLD, Font.ITALIC,
                                                   Font.BOLD + Font.ITALIC};
      int ty = 320;
      for (int i = 0; i < modi.length; i++)
         g.drawString(modi[i], 110 + 50*i, ty-15);
      FontMetrics fm = g.getFontMetrics();
      for (int i = 0; i < fonti.length; i++)
         g.drawString(fonti[i], 108 - fm.stringWidth(fonti[i]), ty + 12*i);
      Font cf = g.getFont();
      for (int i = 0; i < fonti.length; i++, ty += 12)
         for(int j = 0; j < modi.length; j++) {
            g.setFont(new Font(fonti[i], imodi[j], 10));
            g.drawString(modi[j], 110 + 50*j, ty);
         }
   }
   public static void main(String argv[]) {
      new F17pdf088();
   }
}
```

La pagina come è vista da Adobe Acrobat Reader è riprodotta nella figura 6.6.
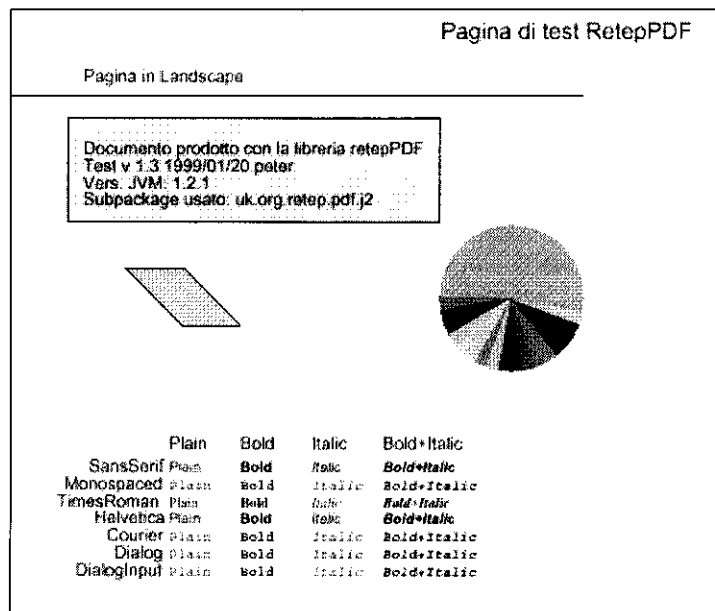
Figura 6.6 Pagina in formato PDF prodotta in Java con la libreria *retepPDF*.

2.3 Comunicazioni (*I/O*) sulle porte in Java

La libreria di classi per le comunicazioni in Java (**Java Communications API**), costituisce un'estensione standard di Java contenuta nel package **javax.comm**, che consente di scrivere applicazioni di comunicazione indipendenti dalla piattaforma e offre il supporto per l'input e l'output (asincrono e sincrono) sulle porte di comunicazione seriali di tipo RS-232 e parallele IEEE 1284, così che è possibile per esempio far comunicare (trasmettere e ricevere dati) attraverso una porta seriale un computer con un dispositivo periferico esterno, che può essere un modem, un fax, una stampante, un lettore di codice a barre o di smartcard o altro dispositivo seriale. Questa API permette inoltre la ricezione di eventi relativi al cambiamento di stato delle dette porte: per esempio se una porta seriale subisce un cambiamento di stato dovuto a *"Carrier Detect"* l'oggetto `SerialPort` propaga un evento di tipo `SerialPortEvent` che caratterizza il cambiamento di stato.

Questa libreria fa ricorso a concetti classici Java come gli stream, per leggere e scrivere sulle porte, e il modello Java di eventi, per rilevare lo stato dei dati sulle porte.

Se si vuol sapere se sono arrivati nuovi dati sul buffer di input si può procedere in due modi: andando ad esaminare periodicamente il buffer per verificare se ci sono nuovi dati (*polling*), oppure attendere l'evento di presenza di nuovi dati nel buffer perchè un nuovo dato invia la notificazione o evento (*listening*).

Se si ha a disposizione un nuovo dispositivo da collegare al computer bisogna consultare il manuale proprio del dispositivo per ciò che riguarda il protocollo di connessione a una porta seriale.

Dal punto di vista del programma bisogna aggiungere, condizionare e aprire il dispositivo.

Esempio, tratto dalla documentazione dell'API stessa, che apre una porta seriale e crea un thread in cui vengono letti i dati su quella porta in seguito ad un *event callback*:

```
// F18leggeCom084.java (API) Esempio di lettura dala porta seriale

import java.io.*;
import java.util.*;
import javax.comm.*;

public class F18leggeCom084 implements Runnable, SerialPortEventListener {

   static CommPortIdentifier portId;
   static Enumeration portList;
   InputStream is;
   SerialPort sp;
   Thread th;

   public static void main(String[] args) {
      portList = CommPortIdentifier.getPortIdentifiers();
      while (portList.hasMoreElements()) {
         portId = (CommPortIdentifier) portList.nextElement();
         if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
            if (portId.getName().equals("COM1")) {
               leggeCom084 reader = new leggeCom084();
            }
      }
   }
  public leggeCom084() {
                  try {
     sp = (SerialPort) portId.open("SimpleReadApp", 2000);
```

```
                    } catch (PortInUseException e) {}
                    try {
      is = sp.getInputStream();
                    } catch (IOException e) {}
                    try {
      sp.addEventListener(this);
                    } catch (TooManyListenersException e) {}
      sp.notifyOnDataAvailable(true);
                    try {
      sp.setSerialPortParams(9600, SerialPort.DATABITS_8,
                                   SerialPort.STOPBITS_1,
                                   SerialPort.PARITY_NONE);
                   } catch (UnsupportedCommOperationException e) {}
      th = new Thread(this);
      th.start();
   }
   public void run() {
      try { Thread.sleep(20000); } catch (InterruptedException e) {}
   }
   public void serialEvent(SerialPortEvent event) {
      switch(event.getEventType()) {
         case SerialPortEvent.BI:
         case SerialPortEvent.OE:
         case SerialPortEvent.FE:
         case SerialPortEvent.PE:
         case SerialPortEvent.CD:
         case SerialPortEvent.CTS:
         case SerialPortEvent.DSR:
         case SerialPortEvent.RI:
         case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
            break;
         case SerialPortEvent.DATA_AVAILABLE:
            byte[] buf = new byte[20];
                              try {
            while (is.available() > 0) {
               int numByte = is.read(buf);
            }
                              } catch (IOException e) {}
            System.out.print(new String(buf));
            break;
      }
   }
}
```

Un altro esempio ottenuto dalla stessa fonte apre una porta seriale per scrivervi dei dati:

```
// F19scriveCom085.java (API) Esempio di scrittura sulla porta seriale

import java.io.*;
import java.util.*;
import javax.comm.*;

public class F19scriveCom085 {

   static Enumeration portList;
   static CommPortIdentifier portId;
   static String messageString = "Hello, world!\n";
   static SerialPort sp;
   static OutputStream os;

   public static void main(String[] s) {
      portList = CommPortIdentifier.getPortIdentifiers();
      while (portList.hasMoreElements()) {
```

```
          portId = (CommPortIdentifier) portList.nextElement();
          if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
             if (portId.getName().equals("COM1")) {
                           try {
                sp = (SerialPort) portId.open("SimpleWriteApp", 2000);
                           } catch (PortInUseException e) {}
                           try {
                os = sp.getOutputStream();
                           } catch (IOException e) {}
                           try {
                sp.setSerialPortParams(9600, SerialPort.DATABITS_8,
                                   SerialPort.STOPBITS_1,
                                   SerialPort.PARITY_NONE);
                           } catch (UnsupportedCommOperationException e) {}
                           try {
                os.write(messageString.getBytes());
                           } catch (IOException e) {}
             }
       }
    }
}
```


[] Rinaldo Di Giorgio, Shivaram H.Misore, Java gets Serial Support with the new javax.comm package, JavaWorld, Vol.3, Issue 5, May 1998.

[] Chuck McManis, Opening up new Ports to Java with javax.comm, JavaWorld, Vol.3, Issue 9, September 1998.

3. Librerie standard di Java

3.1 Libreria di sistema e package di classi

L'interfaccia per la programmazione delle applicazioni (*Application Progr. Interf.*, in breve **API**) Java, resa pubblica da Sun, mette a disposizione del programmatore una libreria di classi predefinite (package `java`) suddivisa in nove (sotto)package, ciascuno dei quali comprende una serie di classi ed interfacce. Queste classi di sistema di Java sono quelle che devono essere sempre presenti in un ambiente Java. Si riporta nella tabella seguente l'elenco dei package con alcune classi di esempio per ciascuono di essi.

`java.lang`

contiene le classi proprie del linguaggio, che comprendono la classe `Object`, la classe `String`, la classe `System` e le classi inviluppo (*wrapper*) dei tipi di base, oltre alla classe `Math` con i suoi metodi matematici

`java.io`

contiene le numerose classi per l'input e l'output su stream e file

`java.util`

contiene le classi di utilità come `Vector`, `Hashtable` e `Date`

`java.awt`

contiene le classi per la grafica e le interfacce grafiche utente (GUI) comprendenti `Window`, `Dialog`, `Button`, `Menu`, etc.

`java.net`

contiene le classi per le comunicazioni in rete, tra cui `Socket`, `ServerSocket` e `URL`

`java.applet`

contiene le classi per le applet, principalmente la classe `Applet`.

3.2 Package `java.lang`

3.2.1 Package di default

Tutte le classi contenute nel package **`java.lang`** sono sempre automaticamente importate in ogni programma Java, anche in assenza in esso di una specifica istruzione di `import`.

3.2.2 Classe `String`

La classe **`String`** rappresenta le stringhe di caratteri. Le istanze di questa classe sono costanti: per usare stringhe variabili si deve ricorrere alla classe `StringBuffer` che rappresenta invece stringhe modificabili. Pur tuttavia operazioni sulle stringhe della classe `String` possono essere fatte aggiungendo una stringa ad un'altra con l'operatore `+` o il `+=`,

ma in tal modo non è la stringa originaria che viene modificata, ma viene invece ogni volta creata una nuova stringa.

Tale classe è definita `final` che significa che non se ne possono creare delle sottoclassi.

Ogni volta che si scrive un letterale stringa nel codice viene ad essere creata un'istanza di questa classe.

Costruttori:

String()	crea una stringa vuota (senza caratteri)
String(String)	crea una stringa con gli stessi caratteri di un'altra stringa data
String(char[])	crea una stringa partendo da un dato array di caratteri
String(char[], i, n)	crea una stringa di `n` caratteri da un dato array di caratteri partendo dal carattere `i`-esimo di questo

Metodi principali:

int length()	restituisce la lunghezza in caratteri della stringa
substring(i, f)	crea una nuova stringa come sottostringa dal carattere `i`-esimo al carattere `f`-esimo della stringa originaria
char charAt(i)	restituisce il carattere `i`-esimo della stringa
getChars(i,f,c[],n)	copia i caratteri dall'`i`-esimo all'`f`-esimo della stringa in un array di caratteri `c[]` a partire dalla posizione `n`-esima in esso
int indexOf(int ch)	restituisce la posizione del primo carattere della stringa avente un valore `ch` dato come intero
int lastIndexOf(int ch)	restituisce la posizione dell'ultimo carattere della stringa avente un valore `ch` dato come intero
String concat(String)	crea una nuova stringa aggiungendo una stringa alla stringa originaria
int compareTo(String)	confronta la stringa presente con un'altra (zero se uguali)
String replace(int co, int cn)	crea una nuova stringa dalla presente sostituendovi tutti i caratteri di valore `co` con `cn`
boolean startsWith(char)	verifica se la stringa comincia con un dato carattere
boolean endsWith(char)	verifica se la stringa termina con un dato carattere
String toLowerCase()	crea una nuova stringa dalla presente sostituendovi tutti i caratteri maiuscoli in caratteri minuscoli
String toUpperCase()	crea una nuova stringa dalla presente sostituendovi tutti i caratteri minuscoli in caratteri maiuscoli
char[] toCharArray()	converte la stringa in un array di caratteri

`String valueOf(Object)`	restituisce la stringa che rappresenta un dato oggetto
`String valueOf(char ch[])`	crea una stringa da un array di caratteri dato
`String valueOf(tipo base)`	vari metodi overloaded per creare una stringa da diversi tipi di variabili elementari (`boolean, char, int, long, float, double`)
`String trim()`	crea una nuova stringa dalla presente togliendo gli spazi bianchi alle estremità

Facciamo un esempio in cui si vede come ad un letterale `String` può essere applicato un metodo della classe `String` perché esso rappresenta un oggetto già istanziato:

```
// G01string.java (F.Spagna) Stringa istanziata come letterale

public class G01string {

   public static void main(String a[]) {
      System.out.println("stringa istanziata come letterale".toUpperCase());
   }
}
```

Il risultato è:

```
STRINGA ISTANZIATA COME LETTERALE
```

Riportiamo qui di seguito il risultato di un programma dimostrativo sui valori di ritorno di diversi metodi della classe `String`: come base si è usata una stringa avente uno spazio intermedio ed uno finale: nella prima colonna è mostrata ciascuna espressione nella forma in cui è stata programmata e nella seconda il risultato da essa restituito.

Sorgente#

Aggiungere valueOf con char array e toUpperCase e toLowerCase#

char c[] = { 'a', 'b', 'c' };#

String s = new String(c);#

toLowerCase();#

toUpperCase();#

valueOf(char o array)#

```
String s = "0123456789 Fer ";
 s                              = 0123456789 fer
 s.substring(2,6)               = 2345
 s.substring(4)                 = 456789 fer
```

```
s.charAt(5)                        = 5
s.compareTo("ab")                  = -49
s.concat("spa")                    = 0123456789 fer spa
s.endsWith("er ")                  = true
s.hashCode()                       = 890328950
s.indexOf('5')                     = 5
s.lastIndexOf('6')                 = 6
s.replace('0','1')                 = 1123456789 fer
s.startsWith("012")                = true
s.toCharArray()                    = 0123456789 fer
s.toString()                       = 0123456789 fer
s.trim()                           = 0123456789 fer
s.valueOf(true)                    = true
s.valueOf(2)                       = 2
s.valueOf(3.)                      = 3.0
s.valueOf(new Integer(2))          = 2
String.valueOf(new Integer(2))     = 2
String.valueOf(new Float(3.))      = 3.0
String.valueOf(new Object())       = java.lang.Object@1ee76b
s + "spa"                          = 0123456789 fer spa
s += "spa"                         = 0123456789 fer spa
s (nuovo valore dopo preced.)      = 0123456789 fer spa
```

Esempio di un'espressione che rende maiuscola l'iniziale di una stringa:

```
s.substring(0, 1).toUpperCase() + substring(1)
```

Esempio in cui è mostrato che le modifiche apportate all'interno di un metodo ad una stringa passata al metodo come argomento non si ripercuotono all'esterno del metodo stesso, mentre invece le modifiche possono essere riportate all'esterno se la stringa è restituita dal metodo come valore di ritorno:

```
// G03string.java (F.Spagna) Classe String per stringhe invariabili

public class G03string {

   String s = "Francesco Parugi";
   F01string() {
      String st = trasforma(s);
      System.out.println(s);                               // s non e' cambiato
      System.out.println(st);                           // valore di s trasformato
   }
   String trasforma(String s) {
      s = s.replace('o', 'a').toUpperCase();     // crea nuova stringa s locale
      return s;                      // e ne fa restituire la referenza al metodo
   }
   public static void main(String a[]) {
      new G01string();
   }
}
```

Risultato:

```
Francesco Parugi
FRANCESCA
```

Un anagramma:

```
// G04anagramma.java (F.Spagna) Un anagramma

public class G04anagramma {

   public static void main(String a[]) {

      String s1 = "Francesco Parugi";
      int i[] = { 0, 2, 12, 14, 5, 1, 10, 6, 8, 9, 11, 12, 3, 4, 14, 7 };
      String s2 = "";
      for (int n = 0; n <i.length; n++)
         s2 += s1.charAt[i[n]];
      System.out.println(s2);
   }
}
```

Esempio:

```
// G05string.java (F.Spagna) String introd.come letterali o istanziate col new

public class G05string {

   public static void main(String a[]) {
      String s1 = "nyuszi";
      String s2 = "nyuszi";
      String s3 = new String("nyuszi");
      System.out.println("uguali gli oggetti  s1 e s2?  -" + s1.equals(s2));
      System.out.println("uguali le referenze s1 e s2?  -" + (s1 == s2));
      System.out.println("uguali gli oggetti  s1 e s3?  -" + s1.equals(s3));
      System.out.println("uguali le referenze s1 e s3?  -" + (s1 == s3));
   }
}
```

Risultato:

```
uguali gli oggetti  s1 e s2?  -true
uguali le referenze s1 e s2?  -true
uguali gli oggetti  s1 e s3?  -true
uguali le referenze s1 e s3?  -false
```

3.2.3 Classe `StringBuffer`

La classe **`StringBuffer`** rappresenta stringhe di caratteri che, contrariamente a quelle della classe `String`, possono essere modificate.

Ogni `StringBuffer` ha una determinata capacità di partenza in numero di caratteri (inizialmente 16) e finchè le operazioni di modifica dell'oggetto non provocano il superamento di questa capacità non vengono fatte nuove allocazioni di memoria, se invece il numero di

caratteri oltrepassa la capacità viene fatta automaticamente una nuova allocazione per contenerli.

Costruttori:

StringBuffer()	crea uno `StringBuffer` senza caratteri e capacità 16
StringBuffer(int)	crea uno `StringBuffer` senza caratteri e capacità data
StringBuffer(String)	crea uno `StringBuffer` da una stringa e capacità 16

Metodi principali:

int length()	restituisce la lunghezza della stringa
char charAt(int)	restituisce il carattere ad un certa posizione (indice)
getChars(i,f,c[],i)	copia i caratteri dall'`i`-esimo all'`f`-esimo dello `StringBuffer` in un array di caratteri `c[]` a partire dalla posizione `i`-esima in esso (come quello di `String`)
append(String)	aggiunge una stringa allo `StringBuffer`
insert(int, String)	inserisce una stringa a partire da una certa posizione
toString()	restituisce la stringa come oggetto della classe `String`
StringBuffer reverse()	restituisce uno `StringBuffer` con l'ordine dei caratteri invertito

Esempio di applicazione dei metodi di questa classe:

```
// G06StringBuffer.java  F.Spagna  Esempio di StringBuffer

public class G06StringBuffer {

   public static void main(String a[]) {

      StringBuffer sb = new StringBuffer("stringa");
      System.out.println("contenuto: \"" + sb.toString() + "\"");
      System.out.println("lunghezza: " + sb.length());
      System.out.println("append:    " + sb.append(" A"));
      System.out.println("insert:    " + sb.insert(0, "I "));
      System.out.println("reverse:   " + sb.reverse());
      for (int n = 0; n < sb.length(); n++)
         System.out.println("charAt(" + n + "): " + sb.charAt(n));
      }
}
```

I risultati del programma sono:

```
contenuto: "stringa"
lunghezza: 7
append:    stringa A
insert:    I stringa A
reverse:   A agnirts I
charAt(0): A
charAt(1): 
charAt(2): a
charAt(3): g
charAt(4): n
charAt(5): i
charAt(6): r
charAt(7): t
charAt(8): s
charAt(9): 
charAt(10): I
```

3.2.4 Classe `StringTokenizer`

La classe **`StringTokenizer`** permette di suddividere una stringa in varie parti (*tokens*) secondo la specificazione di una serie di caratteri delimitatori che separano i vari pezzi.

Ci sono due comportamenti possibili secondo che il flag `returnTokens` è posto uguale a `false` o `true`: nel primo caso i delimitatori stessi non fanno parte dei token, nel secondo caso costituiscono anch'essi stessi dei token.

Costruttori:

`StringTokenizer(String s,String d,boolean)`
crea uno `StringTokenizer` per una stringa data `s`, specificando i delimitatori `d` e precisando se i delimitatori stessi devono o no essere mantenuti come token

Metodi principali:

`nextToken()`	quando usato ripetutamente restituisce ogni volta il prossimo token
`hasMoreTokens()`	restituisce `true` se ci sono ancora token via via che questi sono ricercati con `nextToken()`

E vedi anche esempio di Elliotte.#

Ecco un esempio di applicazione di questa classe con diversi argomenti del costruttore:

```
// G06tokenizer.java (F.Spagna) Esempio di StringTokenizer
```

```
import java.util.*;

class G06tokenizer {

   static public void main (String s[]) {

      StringTokenizer st = new StringTokenizer("stringa da suddividere");
      while (st.hasMoreTokens())
         System.out.println(st.nextToken());

      st = new StringTokenizer("stringa&da&spezzare", "&", false);
      System.out.println("\n      numero di token: " + st.countTokens());
      while (st.hasMoreTokens())
         System.out.println("      " + st.nextToken());

      st = new StringTokenizer("stringa&da&spezzare", "&", true);
      System.out.println("\n           numero di token: " + st.countTokens());
      while (st.hasMoreTokens())
         System.out.println("           " + st.nextToken());
   }
}
```

Il risultato è:

```
stringa
da
suddividere

      numero di token: 3
      stringa
      da
      suddividere

           numero di token: 5
           stringa
           &
           da
           &
           suddividere
```

3.2.5 Classi inviluppo di tipi di dati di base

3.2.5.1 Classi per i tipi di base

I tipi di dati di base, come gli `int`, i `float`, i `boolean`, e gli altri, possono essere rappresentati anche come oggetti di classi particolari predisposte per incapsularli e dare loro certe funzionalità intrinseche a livello di classe (si parla perciò di una loro "promozione allo stato di oggetti"). Spesso i metodi della libreria di Java prevedono come argomenti tali tipi di oggetti, piuttosto che i semplici tipi di base.

Esistono quindi la classe `Long`, la classe `Float`, la classe `Byte`, la classe `Boolean`, e così via, aventi lo stesso nome del tipo primitivo corrispondente, ma convenzionalmente con l'iniziale maiuscola (salvo per `int` che ha il suo corrispettivo nella classe chiamata `Integer`).

Per creare un oggetto di una di queste classi si può usare l'operatore `new` ed un costruttore che riceve il valore del tipo di base come argomento. Ad esempio:

```
enne = new Integer(n);
f = new Float(3.1);
```

La classe **`Integer`** e la classe **`Long`** incapsulano il valore di un tipo primitivo rispettivamente `int` o `long` e forniscono ad essi una certa funzionalità mediante diversi metodi di cui una certa parte metodi di classe, che possono quindi essere usati e consentire varie operazioni su di essi anche senza una specifica istanziazione di oggetti.

La classe **`Float`** e la classe **`Double`** incapsulano il valore di un tipo primitivo rispettivamente `float` e `double`.

Per ricavare il valore in virgola mobile rappresentato a caratteri in una stringa si può scrivere:

```
float v = Float.valueOf("5.5").floatValue();
```

che dà il valore di 5.5 in `float` (analogamente per `double`).

La classe **`Boolean`** incapsula il valore di un tipo primitivo `boolean`.

La classe **`Character`** incapsula il valore di un tipo primitivo `char`.

3.2.6 Classe `Math`

3.2.6.1 Funzioni matematiche come metodi della classe

La classe **`Math`**, definita come `final`, e quindi non derivabile, contiene diversi metodi che calcolano varie funzioni matematiche elementari. Tale classe non deve mai essere istanziata.

Essa contiene come variabili di classe (`static`) il valore di pi greca (`Math.PI`) e quello di e, base dei logaritmi naturali (`Math.E`).

Variabili di classe:

`PI`	valore di tipo `double` di π, rapporto tra circonferenza e diametro di un cerchio
`E`	valore di tipo `double` di e, base dei logaritmi naturali

Metodi:

`sqrt(double)`	restituisce la radice quadrata di un numero
`exp(double)`	restituisce **e** elevato all'argomento
`pow(double a,double b)`	restituisce la potenza `b`-esima di `a`
`log(double)`	restituisce il logaritmo naturale (in base **e**) di un `double`
`sin(double)`	restituisce il seno di un angolo dato in radianti
`cos(double)`	restituisce il coseno di un angolo dato in radianti

tan(double)	restituisce la tangente di un angolo dato in radianti
asin(double)	restituisce l'arcoseno (in radianti) di un numero
acos(double)	restituisce l'arcocoseno (in radianti) di un numero
atan(double)	restituisce l'arcotangente (in radianti) di un numero
atan2(double,double)	restituisce l'angolo theta per la conversione da coordinate rettangolari (x, y) in coordinate polari (r, θ)
ceil(double)	restituisce il più piccolo valore `double` non inferiore all'argomento ed uguale ad un intero
floor(double)	restituisce il più grande valore `double` non superiore all'argomento ed uguale ad un intero
rint(double)	restituisce l'intero (`int`) più vicino all'argomento `double`
round(float)	restituisce il valore `int` più vicino all'argomento `float`
round(double)	restituisce il valore `long` più vicino all'argomento `double`
random()	restituisce un numero a caso (random) compreso tra 0.0 e 1.0
abs(num)	vari metodi overloaded che restituiscono il valore assoluto di un numero che può essere di tipo `int, long, float o double`
max(a, b)	vari metodi overloaded che restituiscono il valore massimo tra due valori numerici dello stesso tipo (due `int`, due `float`, etc.)
min(a, b)	vari metodi overloaded che restituiscono il valore minimo tra due valori numerici dello stesso tipo (due `int`, due `float`, etc.)

E' qui riportata una semplice applicazione grafica che adopera i metodi della classe `Math` per tracciare le curve relative a varie funzioni matematiche:

```
// G07math.java (F.Spagna) Metodi della classe Math

import java.awt.*;

public class G07math extends java.applet.Applet {

   static int H = 60, NCURVE=7, NPUNTI=315;
   double y[][] = new double[NCURVE][NPUNTI];

   public void init() {
      for (int c = 0; c < NCURVE; c++)
         for (int p = 0; p < NPUNTI; p++) {
            double f = 0.;
            switch (c) {
               case 0 : f = Math.sin (p*0.01);      break;
               case 1 : f = Math.cos (p*0.01);      break;
               case 2 : f = Math.tan (p*0.01);      break;
               case 3 : f = Math.sqrt(p*0.01);      break;
               case 4 : f = Math.exp (p*0.01);      break;
               case 5 : f = Math.log (p*0.01);      break;
               case 6 : f = Math.pow (p*0.01, 1.5); break;
            }
            y[c][p] = H * (2.5 - f);
         }
```

```
    }
    public void paint(Graphics g) {
      String s[] = { "0","1","2","3",
                     "sin","cos","tan","tan","sqrt","exp","log","pow(x,1.5)" };
      int a[] = {    3, 97,197,297,172,183,80,220,230,39, 40,177 }; //posizioni
      int b[] = { 164,164,164,164, 88,186,60,250, 51,40,220, 20 };//di scritte
      g.setColor(Color.orange);
      g.fillRect(  0, (int)(H*1.5), 100, H);
      g.fillRect(200, (int)(H*1.5), 100, H);
      g.setColor(Color.black);
      g.drawLine(0, (int)(H*2.5), NPUNTI, (int)(H*2.5));
      for (int n = 0; n < s.length; n++)
         g.drawString(s[n], a[n], b[n]);
      for (int c = 0; c < NCURVE; c++)
         for (int p = 1; p < NPUNTI; p++)
            g.drawLine(p-1, (int)y[c][p-1], p, (int)y[c][p]);
    }
}
```

La figura 7.1 mostra le curve ottenute con questo programma:
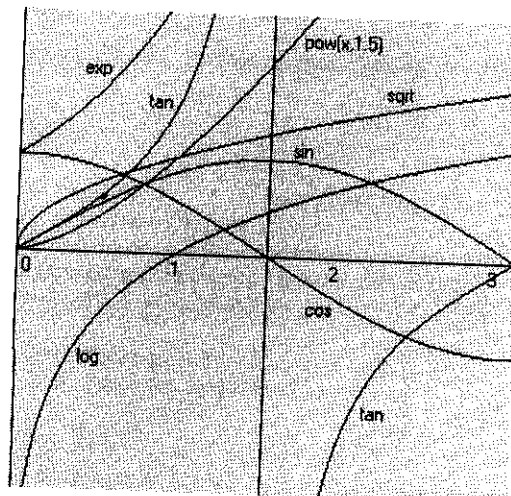
Figura 7.1 Curve di varie funzioni matematiche con i metodi della classe `Math`.

3.2.6.2 Numeri a caso (random)

Il metodo **`Math.random()`** che genera numeri a caso usa come seme il valore del tempo attuale in microsecondi e quindi non richiede che sia fatta esplicitamente all'inizio un'operazione di *sead*. Se si vuole usare un seme diverso si può utilizzare comunque il metodo `java.util.Random()` che riceve il seme come argomento.

3.2.7 Classe `Class`

La classe **`Class`** rappresenta le classi e le interfacce presenti in un'applicazione Java. Anche gli array sono istanze di questa classe, così come anche i tipi primitivi e finanche la parola `void`.

La classe di un determinato oggetto `obj` può essere ricavata con il metodo:

```
obj.getClass()
```

e quindi il nome della classe con:

```
obj.getClass().getName()
```

Metodi:

forName(String)	restituisce l'oggetto `Class` delle classe di un dato nome
getSuperclass()	restituisce l'oggetto `Class` della superclasse di una data classe

Viene fatto qui un esempio di applicazione dei metodi della classe `Class` per trovare tutte le classi ascendenti della classe di un determinato oggetto, in particolare di un oggetto di classe `java.applet.Applet`:

```
// G08class.java (F.Spagna) Lista delle superclassi di un oggetto

public class G08class {

   G07class(Object ogg) {
      System.out.println("oggetto:  " + ogg.toString());
      Class ultima = ogg.getClass();
      System.out.println("classe dell'oggetto: " + ultima.toString());
      while ((ultima = ultima.getSuperclass()) != null)
         System.out.println("classe madre:        " + ultima.toString());
   }
   public static void main(String[] v) {
      new G08class(new java.applet.Applet());
   }
}
```

Il risultato è:

```
oggetto: java.applet.Applet[panel0,0,0,0x0,invalid,layout=java.awt.FlowLayout]
classe dell'oggetto: class java.applet.Applet
classe madre:        class java.awt.Panel
classe madre:        class java.awt.Container
classe madre:        class java.awt.Component
classe madre:        class java.lang.Object
```

3.3 Package java.util

3.3.1 Vector

La classe **Vector** del package `java.util` rappresenta un array di oggetti in numero che può aumentare o diminuire quanto si vuole. Un oggetto di questa classe non può contenere elementi costituiti da variabili di tipo elementare, ma solo oggetti (istanze di classi), e basta che siano oggetti senza che debbano necessariamente appartenere alla stessa classe, come è invece richiesto per gli array. Nel caso si volessero adoperare dei tipi elementari in un `Vector` bisognerebbe, al momento del loro inserimento nel `Vector`, trasformarli prima in oggetti, per esempio per un `int` in un tipo `Integer`, e ritrasformarli poi in tipo elementare quando devono essere usati singolarmente.

L'introduzione di nuovi elementi in un oggetto di questa classe viene fatta con con il metodo `add(Object)` o l'eliminazione di elementi esistenti con `remove()`.(quadro metodi#)

Come negli array gli elementi di un `Vector` sono accessibili mediante un indice intero. Una certa ottimizzazione della gestione della memoria viene fatta con il concetto di capacità (*capacity*).

3.3.2 Hashtable

La classe **Dictionary** del package `java.util` è una classe astratta madre di classi che permettono di mappare una serie di valori con delle chiavi.

La classe **Hashtable** del package `java.util`, sottoclasse di `Dictionary`, rappresenta una mappatura di una serie di oggetti, detti **valori**, con un'altra serie di oggetti detti **chiavi**.

La classe `Hashtable` non accetta tipi primitivi, come per esempio `int`, né come elementi da mappare nè come chiavi, perché i suoi elementi devono essere degli oggetti.

Un esempio potrebbe essere quello che facciamo qui di seguito consistente in una mappatura di una serie di colori (definiti come oggetti di classe `java.awt.Color`) con i loro nomi rappresentati da oggetti della classe `java.lang.String` in funzione di chiavi:

```
// G09hash.java (F.Spagna) Esempio di put() e get() in un Hashtable

mport java.util.*;
import java.awt.*;

class G09hash {

   Color col[] = { Color.red, Color.yellow, Color.green, Color.blue };
   String nom[] = { "rosso", "giallo", "verde", "blu" }
   Hashtable colori = new Hashtable();

   G09hash() {
      for (int n = 0; n < col.length; n++)
         colori.put(nom[n], col[n]);
      Color c = (Color)colori.get("rosso");
      if (c != null)                           // se trova una corrispondenza
         System.out.println("colore " + c);
```

```
    }
}
```

Risultato#

Come si vede nell'esempio, per risalire dalla chiave (un oggetto `String`) all'oggetto corrispondente (un oggetto `Color`) si usa il metodo `get()` e si deve fare un casting del valore da esso restituito verso la classe del valore cercato, perché `get()` restituisce semplicemente un'istanza della classe `Object`.

Costruttori:

#

Metodi:

3.3.3 `Properties`

La classe **`Properties`**, derivata da `Hashtable`, ha in più della classe genitrice la capacità di persistenza che permette ai suoi oggetti di essere salvati in uno stream o caricati da uno stream. Per queste operazioni la classe dispone dei due metodi:

`load(InputStream)`	legge una lista di coppie chiavi ed elementi da un InputStream
`store(OutputStream,String)`	scrive una lista di coppie chiavi ed elementi ad un OutputStream

3.4 Date e ore

3.4.1 La classe `Date`

Programma che legge in continuazione dal sistema operativo il tempo e lo affigge sullo schermo.

```
// G10orologio.java (F.Spagna) Un orologio digitale

import java.awt.*;                                    // per Graphics e Color
import java.util.Date;                                               // per ora

public class G10orologio extends java.applet.Applet implements Runnable {

   Label lab = new Label("00.00.00");
   Thread Th;

   public void init() {
      lab.setFont(new Font("", Font.BOLD, 28));
      add (lab);
   }
```

```
    public void start() {
       if (Th == null) {
          Th = new Thread(this);
          Th.start();
       }
    }
    public void run() {
       while (Th != null) {
       Date adesso = new Date();
       String orario = "";
       int ora = adesso.getHours();
       int min = adesso.getMinutes();
       int sec = adesso.getSeconds();
       if (ora < 10) orario += "0";
       orario += (ora + ":");
       if (min < 10) orario += "0";
       orario += (min + ":");
       if (sec < 10) orario += "0";
       orario += sec;
       lab.setText(orario);
          try { Th.sleep(1000); } catch (InterruptedException e){}
       }
    }
    public void stop() {
       Th.stop();
       Th = null;
    }
    public void update(Graphics g) {
       paint(g);
    }
}
```

In figura 7.2 è mostrata l'applet catturata ad un istante durante il funzionamento.
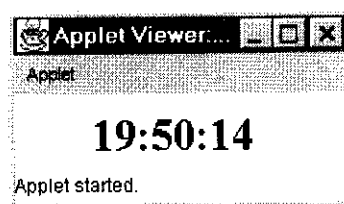
Figura 7.2 Esempio di simulazione di un orologio digitale.

3.4.2 Localizzazione di date e ore

Le date e le ore possono essere presentate in formati diversi secondo modalità dipendenti dal paese. Ecco un esempio:

```
// G11dataora.java (F.Spagna) Vari formati locali per le date e le ore

import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

public class G11dataora {

   public static void main (String args[]) throws java.io.IOException {
```

```
        Date d = new Date();
        System.out.println(
             DateFormat.getDateInstance().format(d)
      +"\n"+ DateFormat.getDateInstance(DateFormat.FULL,Locale.ITALY).format(d)
      +"\n"+ DateFormat.getDateInstance(DateFormat.FULL,Locale.FRANCE).format(d)
      +"\n"+ DateFormat.getDateInstance(DateFormat.FULL,Locale.US).format(d)
      +"\n"+ DateFormat.getDateTimeInstance(DateFormat.SHORT, DateFormat.LONG,
                                            Locale.FRANCE).format(d)
      +"\n"+ DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.MEDIUM,
                                            Locale.UK).format(d)
        );
    }
}
```

I risultati sono mostrati qui di seguito:

```
19-dic-98
sabato 19 dicembre 1998
samedi 19 décembre 1998
Friday, December 18, 1998
19/12/98 03:07:03 GMT+01:00
19-Dec-1998 02:07:03
```

TOGLIERE?#

3.5 Tavola riassuntiva di package e classi

java.applet	contiene la classe `Applet`, superclasse di ogni applet
interface AudioClip	interfaccia per audio
class Applet	superclasse di ogni applet

java.io	contiene le classi per l'input e l'output
class	
File	file
InputStream	flusso di input
OutputStream	flusso di output
DataInput	ff
DataOutput	ff
DataInputStream	ff
DataOutputStream	ff
FileInputStream	ff
FileOutputStream	ff
FilterInputStream	ff
FilterOutputStream	ff
FilenameFilter	ff
PrintStream	ff
RandomAccessFile	ff
StreamTokenizer	ff

java.lang	contiene le classi di base per il linguaggio
interface Runnable	ha un metodo `run()`
class	
Boolean	gestisce i `boolean`
Character	gestisce i caratteri
Class	cc
Double	gestisce i `double`
Float	gestisce i `float`
Integer	gestisce gli `int`
Long	gestisce i `long`
Number	nn
Math	contiene tutti i metodi matematici
Object	capostipite di tutte le classi di Java
Process	pp
String	gestisce le stringhe di caratteri
StringBuffer	buffer di caratteri
System	classe del sistema
Thread	thread (flusso di un processo)

java.net	contiene le classi per l'accesso a reti distribuite

SOMMARIO